An Admissible Second Order Frame Rule in Region Logic

David A. Naumann

Stevens Institute of Technology, CS Report 2008-02

April 21, 2008
Slightly revised and reformatted from version circulated in February 2008.

Abstract: Shared mutable objects and reentrant callacks can subvert encapsulation in object-based programs. For modular reasoning, verifiers rely on methodologies. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications, which pose challenges for proving soundness and for comparing/combining methodologies. This paper formalizes a second order frame rule, similar to that of separation logic but for a logic with explicit regions. The rule captures proof obligations of invariant methodologies such as ownership for dynamically instantiable abstractions, using an extended provability judgement that threads a state-based separation discipline through a proof to points where encapsulation is at risk. Soundness with respect to a standard semantics is proved using an admissibility argument for the second order frame rule.

This material is based upon work supported by the
National Science Foundation under grants CNS-0627338, CCF-0429894, and
CRI-0708330.

An Admissible Second Order Frame Rule in Region Logic

David A. Naumann
Stevens Institute of Technology, Hoboken, NJ, USA

April 21, 2008

Abstract

Shared mutable objects and reentrant callacks can subvert encapsulation in object-based programs. For modular reasoning, verifiers rely on methodologies. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications, which pose challenges for proving soundness and for comparing/combining methodologies. This paper formalizes a second order frame rule, similar to that of separation logic but for a logic with explicit regions. The rule captures proof obligations of invariant methodologies such as ownership for dynamically instantiable abstractions, using an extended provability judgement that threads a state-based separation discipline through a proof to points where encapsulation is at risk. Soundness with respect to a standard semantics is proved using an admissibility argument for the second order frame rule.

1 Introduction

An essential technique for reasoning about modular code was articulated by Hoare [Hoa72]: Clients of a module are verified using a public specification, say $\{P\}\ m\ \{P'\}$ where m is a method (procedure) name, but the verification condition for the method's body, B, is $\{P \wedge Q\}\ B\ \{P' \wedge Q\}$. Here Q is the module invariant, supposed not to be susceptible to client interference. One way to justify Hoare's mismatch draws on the rule of Invariance, which in its original form says that $\{P\}\ C\ \{P'\}$ implies $\{P \wedge Q\}\ C\ \{P' \wedge Q\}$ provided that command C writes no variables read in Q. A module construct could ensure, for reasons of scope, this disjointness of variables. Then any proof of a client program, using interface specification $\{P\}\ m\ \{P'\}$, could be transformed to one using the specification $\{P \wedge Q\}\ m\ \{P' \wedge Q\}$ against which B is verified, by adding instances of Invariance to conjoin Q at appropriate points.

In the presence of various language features this simple story is subverted because it is difficult to specify and verify the absence of interference. In particular, verification systems for object-oriented programs, e.g., ESC/Java and Spec#, try to provide sound forms of Hoare's mismatch by imposing programming disciplines. These combine special annotations or types with instrumentation (ghost state) and syntactic restrictions on programs and specifications. For example, the Universe Type system [MPHL06] provides heap encapsulation via a *confinement invariant* that designates certain objects as owned

by others and it *bounds the effects* of client programs, disallowing write access to non-owned objects. The mismatch is then sound for invariants that depend only on owned objects.

This paper studies a logic in which such disciplines can be encoded and deployed on a per-module basis. It supports invariants at varying granularity ranging from a single instance (and its owned representation) to clusters with shared representation, which occur in various important design patterns [LLM07]. The mismatch enabled by a discipline is made explicit in a *second order frame rule* adapted from that of separation logic [OYR04]. The ordinary frame rule of separation logic resembles the Invariance rule, but without using a side condition for absence of interference: it says that $\{P\}\ C\ \{P'\}$ implies $\{P * Q\}\ C\ \{P' * Q\}$ where $P * Q$ says the truth of P and Q is supported by disjoint footprints in the heap. The antecedent $\{P\}\ C\ \{P'\}$ bounds the effects of C; loosely speaking, its footprint is within the confines of P and P'. The second order frame rule lifts this reasoning to the level of assumptions, distilling the essence of Hoare's mismatch.

In separation logic, the second order frame rule has nontrivial interaction with the ordinary rule of conjunctivity, owing to the existential implicit in the semantics of $*$. The soundness result in [OYR04] restricts the rule to *precise* invariants Q, i.e., with uniquely determined footprints. Soundness, without the restriction to precise predicates, has been proved in semantic models that do not validate the rule of conjunctivity [BTSY05, PBNM08].

The logic in this paper uses regions to make footprints explicit. Regions have been used in several calculi to track disjointness of heap effects for various purposes [TT94]. Here, an ordinary first order assertion language is augmented with type **rgn** for finite sets of allocated references, following Kassios [Kas06]. He advocates the use of ghost variables and fields of type **rgn** and a second order predicate, G **frames** Q, that says the footprint of Q in the current state is within the region currently denoted by expression G. Working in a relational calculus of refinement [Heh84], Kassios shows by example how regions support explicit formulation of existing and novel disciplines, without additional machinery and without imposing one discipline on all modules.

Aiming to adapt Kassios' insights to conventional verifiers using specification languages like JML, Banerjee et al [BNR08] introduce a Hoare logic where effects in the "modifies clause" can refer to region expressions. Their logic relies on a subsidiary judgement of the form $P \vdash \overline{\varepsilon}$ **frm** Q that says the read effect or footprint of formula Q is conservatively approximated by the set of effects $\overline{\varepsilon}$, in any state satisfying P (where P, Q are ordinary first order formulas). Their frame rule has an explicit proviso about disjointness, expressed in terms of the footprint of Q and the write effect of the command. The formulas are a mild extension of first-order classical logic logic; there is no separating conjunction per se.

In this paper, their logic is extended with a second order frame rule. The rule relies on a judgement form that serves to thread a confinement invariant and client effect bound — which together embody some discipline for encapsulation— through the part of a program proof in which client code is verified against assumed specifications that do not mention the module invariant. Our main result implies the soundness of this logic with respect to a standard denotational semantics. The second order frame rule is treated indirectly: it is proved to be *admissible*, i.e., any proof can be transformed (in the manner sketched

above) to one not using it.

Outline. Sec. 2 illustrates the use of regions for reasoning in the small, to introduce the partial correctness logic with regions [BNR08] which is reviewed in Sec. 3–5. Sec. 6 illustrates how the second order frame rule is used for reasoning in the large, borrowing some material from Banerjee et al. [BNR08]. Sec. 7 formalizes the extended logic. Its soundness is proved in Sec. 8. Sec. 9 has concluding remarks. Appendices give further details. A shorter version of the paper can be found online at `http://www.cs.stevens.edu/~naumann/pub/hofrl.pdf`.

2 Framing and footprints with explicit regions

Let Γ be the context $x\colon K, y\colon K, r\colon \mathbf{rgn}$ where K is some class with field $f\colon K$. Values of type **rgn** are finite sets of allocated references. Expression $\langle y\rangle$ denotes a singleton region or the empty set, depending on whether y is **null**, and $y \in r$ abbreviates $y \neq \mathbf{null} \wedge \langle y\rangle \subseteq r$. The axiom for field update (plus consequence rule to add $y \notin r$) yields

$$\{y \neq \mathbf{null} \wedge y \notin r\}\ y.f := x\ \{y.f = x\}[\mathbf{wr}\langle y\rangle{\bullet}f] \tag{1}$$

The effect $\mathbf{wr}\langle y\rangle{\bullet}f$ says that the command updates no state other than, possibly, the f field of an object (reference) in $\langle y\rangle$. Primitive formula $r{\bullet}f \subseteq r$ says r is closed under dereferences of field f. (More precisely: every object o in r with field f has $o.f$ in r.) We will show it is preserved by the update, using the rule

$$\textsc{Frame}\quad \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad P \vdash \overline{\delta}\ \mathbf{frm}\ Q \qquad P \Rightarrow \overline{\delta} \star \overline{\varepsilon}}{\{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}]}$$

Like Hoare's rule of invariance, Frame conjoins Q provided that C cannot interfere with Q. Side condition $P \vdash \overline{\delta}\ \mathbf{frm}\ Q$ bounds the footprint of Q to some read effects, $\overline{\delta}$. The operator $\star$ generates a conjunction of region disjointness formulas sufficient to ensure that the writes on the right cannot interfere with the reads on the left, so $P \Rightarrow \overline{\delta} \star \overline{\varepsilon}$ says that footprint $\overline{\delta}$ of Q is disjoint from the writes of C. Antecedent P can be assumed since the regions mentioned in reads and writes are interpreted in the initial state. For the example, $\mathbf{true} \vdash \mathbf{rd}\, r, \mathbf{rd}\, r{\bullet}f\ \mathbf{frm}\ r{\bullet}f \subseteq r$ and the separator formula, $\mathbf{rd}\, r, \mathbf{rd}\, r{\bullet}f \star \mathbf{wr}\langle y\rangle{\bullet}f$ simplifies to $r \# \langle y\rangle$. Now $r \# \langle y\rangle$ just says the two sets are disjoint, which follows from $y \notin r$, so (1) yields $\{y \neq \mathbf{null} \wedge y \notin r \wedge r{\bullet}f \subseteq r\}\ y.f := x\ \{r{\bullet}f \subseteq r\}[\mathbf{wr}\langle y\rangle{\bullet}f]$ by an application of Frame.

As mentioned in Sec. 1, the frame rule in separation logic has no side conditions but rather uses the separated conjunctions $P * Q$ and $P' * Q$. Unlike Frame, this disallows the reads of C to overlap the reads of Q; the issue has been addressed using permissions [BCOP05].

3 Illustrative programming language

The formalization caters for the intended application to Java-like programming languages and specification languages like JML and Spec#, in that we consider typed, first-order

$$
\begin{array}{lll}
T & ::= \mathbf{int} \mid K \mid \mathbf{rgn} & \text{for } K \in \mathit{DeclaredClassName} \\
E & ::= x \mid c \mid \mathbf{null} \mid E \oplus E & \text{for } \oplus \in \{=, +, >, \ldots\},\ c \in \mathbb{Z} \\
G & ::= x \mid x.f \mid \langle E \rangle \mid \mathbf{emp} \mid \mathbf{alloc} \mid G \otimes G & \text{for } \otimes \in \{\cup, \cap, -\} \\
F & ::= E \mid G & \\
M & ::= m(\overline{x}\colon \overline{T}) & \text{for } m \in \mathit{MethodName} \\
C & ::= m(\overline{z}) \mid \mathbf{let}\ \overline{M}\ \mathbf{be}\ \overline{C}\ \mathbf{in}\ C \mid x := F \mid x := \mathbf{new}\ K \mid x := x.f \mid x.f := F & \\
 & \quad \mid \mathbf{if}\ x\ \mathbf{then}\ C\ \mathbf{else}\ C \mid \mathbf{while}\ x\ \mathbf{do}\ C \mid C\,;C \mid \mathbf{var}\ x\colon T\ \mathbf{in}\ C\ \mathbf{end} &
\end{array}
$$

Figure 1: Programming language. Let $x, y, r \in \mathit{VarName}$ and $f, g, h \in \mathit{FieldName}$.

objects. To streamline the logic, however, methods are not bound to classes and only a rudimentary module construct is considered. A simple denotational semantics is used as in [BNR08] and in anticipation of machine checking the results in an existing formalization of a fragment of Java/JML in the PVS theorem prover [LNR06].

Assume we are given a *class table*, i.e., declarations of a number of named record types. Record fields, like method parameters and program variables, are assigned data types: **int**, **rgn**, or class name. Fields of a class may make mutually recursive reference to other classes. We omit concrete syntax for classes; *ClassName* is the set of names of declared classes, fields(K) gives the field declarations for class K. Field names are assumed globally unique.

Syntax. A main program is a command C in the context of some variable and method declarations, written $\Gamma; \overline{M} \vdash C$. Here Γ is a finite map from variable names to data types. Finite maps are also written, e.g., $\overline{x}\colon \overline{T}$ for a list of variable declarations; comma means union of disjoint maps and binds tightly. The signatures of methods in scope are given by the list $\overline{M}$. Parameters are passed by value and methods do not have a result. For the typing judgement $\Gamma; \overline{M} \vdash C$ to be well formed, all free variables of C must be in $\mathrm{dom}(\Gamma)$ and no method name is duplicated in $\overline{M}$. Programs should be typable as usual for Java-like languages (e.g., [LNR06]), disallowing pointer arithmetic.

N.B. We assume the *hygiene condition* that no variable occurs both bound and free.

The grammar is in Fig. 1. As in separation logic, ordinary expressions E do not depend on the heap; command $x := y.f$ is included for reading a field. Region expressions G can have one-step dependence, in the case of $x.g$ where field g has type **rgn**. Effects and formulas involve the form $G\bullet f$ but this is *not* a region expression. Equality test $E = E'$ is only for expressions of reference type or **int** (and equality is the only comparator for references; we have no pointer arithmetic). The only operations on expressions of type **rgn** are $\cup, \cap, -$. These operations all return type **rgn**, so it is clear region variables/fields are only ghosts for reasoning.

The construct **let** $\overline{N}$ **be** $\overline{C}$ **in** C binds a list $\overline{N}$ of method signatures to a same-length list $\overline{C}$ of commands that may make mutually recursive reference to each other, may use global variables, and may be used in the body C. Here is the typing rule:

$$
\frac{\Gamma; \overline{M}, \overline{N} \vdash C \qquad \Gamma, \overline{x}\colon \overline{T};\ \overline{M}, \overline{N} \vdash B_i \text{ where } m(\overline{x}\colon \overline{T}) \equiv \overline{N}_i, \text{ for all } i}{\Gamma; \overline{M} \vdash \mathbf{let}\ \overline{N}\ \mathbf{be}\ \overline{B}\ \mathbf{in}\ C}
$$

$$
\begin{array}{lcl}
[\![r]\!]\sigma & = & \sigma(r) \\
[\![x.g]\!]\sigma & = & \sigma(x.g) \text{ if } \sigma(x) \neq \text{nil, else } \varnothing \\
[\![\langle E \rangle]\!]\sigma & = & \{\, [\![E]\!]\sigma \,\} \text{ if } [\![E]\!]\sigma \neq \text{nil, else } \varnothing \\
[\![\mathbf{emp}]\!]\sigma & = & \varnothing \\
[\![\mathbf{alloc}]\!]\sigma & = & \text{alloc}(\sigma) \\
[\![G_1 \cup G_2]\!]\sigma & = & [\![G_1]\!]\sigma \cup [\![G_2]\!]\sigma \\
[\![G_1 \cap G_2]\!]\sigma & = & [\![G_1]\!]\sigma \cap [\![G_2]\!]\sigma \\
[\![G_1 - G_2]\!]\sigma & = & [\![G_1]\!]\sigma - [\![G_2]\!]\sigma
\end{array}
$$

Figure 2: Semantics, $[\![\Gamma \vdash G\colon \mathbf{rgn}]\!]$, for $g\colon \mathbf{rgn}$.

Method bodies may refer to variables in Γ, which poses no difficulty for the fixpoint semantics since methods are neither stored nor passed as arguments.

An annotated Java program can be modeled as

$$\overline{x}\colon \overline{T}; \varnothing \vdash \mathbf{var}\ \overline{y}\colon \overline{U}\ \mathbf{in\ let}\ \overline{N}\ \mathbf{be}\ \overline{B}\ \mathbf{in}\ C\ \mathbf{end}$$

where the inputs and outputs correspond to variables $\overline{x}$ (none of type **rgn**), all methods in all classes are combined to form $\overline{N}$ and $\overline{B}$, and $\overline{y}\colon \overline{U}$ are the global variables (including those region type). A more sophisticated module construct would allow some globals to be scoped just over $\overline{B}$; this would also provide a place to write specifications of the confinement invariant etc. used in our second order frame rule.

Program semantics. A *state*, σ, consists of an assignment of types to the allocated object references, a heap mapping allocated references to object states, and a store assigning values to the variables in scope.[1] A *state transformer for* Γ is a total function that sends Γ-states to the outcome $\perp$ (for divergence or undefinedness), $\Lightning$ (for runtime errors), or a state. Outcomes are ordered with $\Lightning$ incomparable to states and $\perp$ below all. We say σ' *extends* σ provided $\text{alloc}(\sigma) \subseteq \text{alloc}(\sigma')$ and $\text{type}(o, \sigma) = \text{type}(o, \sigma')$ for all $o \in \text{alloc}(\sigma)$. Commands denote state transformers with the property that the final state, if any, extends the initial one.

A *method environment for* $\Gamma, \overline{M}$ maps each method name m in $\overline{M}$ to a state transformer for $\Gamma, \overline{x}\colon \overline{T}$ where $\overline{x}\colon \overline{T}$ are the parameters of m in $\overline{M}$. The semantics $[\![\Gamma; \overline{M} \vdash C]\!]$ takes a method environment for $\Gamma, \overline{M}$ to a state transformer for Γ, defined by recursion on the structure of C.[3] Details of the semantics are omitted, except for region expressions, see Fig. 2. Null dereference is not an error for region type fields, whereas in the primitive command $x := y.f$ with f not of region type, the program yields $\Lightning$ in case y is null. The semantics is deterministic: Local variables and fields are initialized to fixed default values

[1] We use the following notations: $\sigma(x)$ is the value of x in σ; $\sigma(x.f)$ is the value of field f of object $\sigma(x)$; $\sigma(o.f)$ is the value of f of reference value o; [2] $\text{alloc}(\sigma)$ is the set of all allocated references; $\text{type}(o, \sigma)$ is the allocated type of reference o; $\text{extend}(\sigma, x, v)$ is the state like σ but with variable x (not present in σ) added with value v.

[3] The denotational semantics of commands is conventional. Fixpoints are used for loops and for the recursive "let" construct. If μ is a method environment for $\Gamma; \overline{M}$, then we define $[\![\Gamma; \overline{M} \vdash \mathbf{let}\ \overline{N}\ \mathbf{be}\ \overline{B}\ \mathbf{in}\ C]\!]\mu$ to be $[\![\Gamma; \overline{M}, \overline{N} \vdash C]\!](\mu \oplus \hat{\mu})$ where $\hat{\mu}$ is the least fixed point of the function $\lambda\mu'.\lambda m_i \in \overline{N}.[\![\Gamma, \overline{xi}\colon \overline{Ti};\ \overline{M}, \overline{N} \vdash \overline{B}_i]\!](\mu \oplus \mu')$. Here the ith signature in $\overline{N}$ is supposed to be $m_i(\overline{xi}\colon \overline{Ti})$.

$$\begin{array}{rcl} P, Q, R & ::= & E = E \mid x.f = F \mid G \subseteq G \mid G \# G \mid G{\bullet}f \# G \mid G{\bullet}f \subseteq G \\ & \mid & (\forall x{:}\,\mathbf{int} \mid P) \mid (\forall x{:}\,K \in G \mid P) \mid P \wedge P \mid \neg P \end{array}$$

$\sigma \models x.f = F$	iff	$\sigma(x) \neq \text{nil}$ and $\sigma(x.f) = [\![F]\!]\sigma$
$\sigma \models G_1 \# G_2$	iff	$[\![G_1]\!]\sigma \cap [\![G_2]\!]\sigma = \varnothing$
$\sigma \models G_1{\bullet}f \subseteq G_2$	iff	$\sigma(o.f) = \text{nil}$ or $\sigma(o.f) \in [\![G_2]\!]\sigma$ for all $o \in [\![G_1]\!]\sigma$ with $f \in \text{refields}(o, \sigma)$
$\sigma \models G_1{\bullet}f \# G_2$	iff	$\sigma(o.f) \notin [\![G_2]\!]\sigma$ for all $o \in [\![G_1]\!]\sigma$ with $f \in \text{refields}(o, \sigma)$
$\sigma \models^{\Gamma} \forall x{:}\,K{\in}G \mid P$	iff	$\text{extend}(\sigma, x, o) \models^{\Gamma, x:K} P$ for all o in $[\![G]\!]\sigma$ with $\text{type}(o, \sigma) = K$

Figure 3: Formulas and selected semantics. Using $\text{refields}(o, \sigma)$ for the names of reference-type fields in $\text{type}(o, \sigma)$.

($\varnothing$ for **rgn**) and the semantics of **new** K uses a deterministic allocator, which may be taken to be any function from states to references such that the result is always fresh.

4 Assertions, effects, and separators

The assertion language is first order and does not allow variables of type **rgn** to be bound by quantifiers. The syntax and semantics are given in Fig. 3 (see [BNR08] for typing). It is straightforward to add inductive definitions, as is needed for precise functional specifications. But interesting separation properties can be expressed without them, using the one-step points-to formulas for disjointness ($G{\bullet}f \# G'$) and inclusion ($G{\bullet}f \subseteq G'$). A formula is *valid* if it holds in all states. We write x **is** K to abbreviate $x \neq \mathbf{null} \wedge \mathbf{type}(K, \langle x \rangle)$.

Effects. An *effect set* is written as a comma-separated list, $\overline{\varepsilon}$, of *effects* given by the grammar

$$\varepsilon ::= \mathbf{rd}\, x \mid \mathbf{wr}\, x \mid \mathbf{rd}\, G{\bullet}f \mid \mathbf{wr}\, G{\bullet}f \mid \mathbf{rd}\, \mathbf{alloc} \mid \mathbf{wr}\, \mathbf{alloc} \mid \mathbf{fr}\, G$$

We let $\delta, \varepsilon, \eta$ range over effects. Effects must be well formed (wf) for the context Γ in which they occur: $\mathbf{rd}\, x$ and $\mathbf{wr}\, x$ are wf if $x \in \text{dom}(\Gamma)$; $\mathbf{rd}\, G{\bullet}f$, $\mathbf{wr}\, G{\bullet}f$, and $\mathbf{fr}\, G$ are wf if G is wf in Γ. Effect $\mathbf{wr}\, \mathbf{alloc}$ allows allocation, since **alloc** denotes the region of all currently allocated objects. Effect $\mathbf{fr}\, G$ says that the final value of G contains only freshly allocated objects.

Besides their use for formulas, read effects in method specifications are useful for various purposes. For lack of space in this extended abstract, we omit read effects from the command correctness judgements and rules.[4]

Let effect set $\overline{\varepsilon}$ be well formed in Γ and let σ, σ' be Γ-states. We say $\overline{\varepsilon}$ *allows transition from* σ *to* σ', written $\sigma \to \sigma' \models \overline{\varepsilon}$, iff σ' extends σ and the following all hold:

(T0) for every y in $\text{dom}(\Gamma)$, either $\sigma(y) = \sigma'(y)$ or $\mathbf{wr}\, y$ is in $\overline{\varepsilon}$

(T1) for every $o \in \text{alloc}(\sigma)$ and every $f \in \text{fields}(o, \sigma)$, either $\sigma(o.f) = \sigma'(o.f)$ or there is $\mathbf{wr}\, G{\bullet}f$ in $\overline{\varepsilon}$ such that $o \in [\![G]\!]\sigma$

[4] They are straightforward and not pertinent to our results, with the exception that the substitution rule uses read effects for variables.

$$G_1 \subseteq G_2 \vdash \mathbf{wr}\ G_1{\bullet}f \leq \mathbf{wr}\ G_2{\bullet}f \qquad G_1 \subseteq G_2 \vdash \mathbf{rd}\ G_1{\bullet}f \leq \mathbf{rd}\ G_2{\bullet}f \qquad \mathbf{true} \vdash \overline{\varepsilon} \leq \overline{\varepsilon}, \varepsilon$$

$$\mathbf{true} \vdash \mathbf{wr}\ G_1{\bullet}f, \mathbf{wr}\ G_2{\bullet}f \lesseqgtr \mathbf{wr}(G_1 \cup G_2){\bullet}f \qquad \frac{P \vdash \overline{\varepsilon}_1 \leq \overline{\varepsilon}_2 \qquad P \vdash \overline{\varepsilon}_2 \leq \overline{\varepsilon}_3}{P \vdash \overline{\varepsilon}_1 \leq \overline{\varepsilon}_3}$$

$$\frac{P' \Rightarrow P \qquad P \vdash \overline{\varepsilon} \leq \overline{\varepsilon}'}{P' \vdash \overline{\varepsilon} \leq \overline{\varepsilon}'} \qquad \frac{P \vdash \varepsilon_1 \leq \varepsilon_2}{P \vdash \varepsilon_1, \overline{\varepsilon} \leq \varepsilon_2, \overline{\varepsilon}}$$

Figure 4: Selected sub-effect rules. We write $\lesseqgtr$ to abbreviate two inclusion rules.

(T2) if $\mathrm{alloc}(\sigma') \neq \mathrm{alloc}(\sigma)$ then $\mathbf{wr}\,\mathbf{alloc}$ is in $\overline{\varepsilon}$
(T3) for every $\mathbf{fr}\ G$ in $\overline{\varepsilon}$: $[\![G]\!]\sigma' \subseteq \mathrm{alloc}(\sigma') - \mathrm{alloc}(\sigma)$

Let $\overline{\varepsilon}$ be an effect set and σ, σ' be states such that σ' extends σ. Say that σ and σ' *agree on* $\overline{\varepsilon}$ provided the following hold:

(A0) for all $\mathbf{rd}\,x$ in $\overline{\varepsilon}$, we have $\sigma(x) = \sigma'(x)$
(A1) if $\mathbf{rd}\,\mathbf{alloc}$ in $\overline{\varepsilon}$ then $\mathrm{alloc}(\sigma) = \mathrm{alloc}(\sigma')$
(A2) for all $\mathbf{rd}\ G{\bullet}f$ in $\overline{\varepsilon}$, for all $o \in [\![G]\!]\sigma$ with $f \in \mathrm{fields}(o, \sigma)$, we have $\sigma(o.f) = \sigma'(o.f)$

The judgement $P \vdash \overline{\varepsilon} \leq \overline{\eta}$ expresses that the writes/reads in $\overline{\eta}$ permit more program behaviors than $\overline{\varepsilon}$ does. Fig. 4 gives some of the rules for sub-effects.

Framing. For ordinary expressions, define $\mathrm{ftpt}(E) = \{\mathbf{rd}\,x \mid x \in \mathit{Vars}(E)\}$. For the read effects of region expressions and primitive assertions, $\mathrm{ftpt}(G)$ is defined recursively, e.g.,

$$\begin{array}{ll|ll}
\mathrm{ftpt}(x) & = \{\mathbf{rd}\,x\} & \mathrm{ftpt}(\langle E \rangle) & = \mathrm{ftpt}(E) \\
\mathrm{ftpt}(\mathbf{alloc}) & = \{\mathbf{rd}\,\mathbf{alloc}\} & \mathrm{ftpt}(G_1 \cap G_2) & = \mathrm{ftpt}(G_1) \cup \mathrm{ftpt}(G_2) \\
\mathrm{ftpt}(x.g) & = \{\mathbf{rd}\,x, \mathbf{rd}\langle x \rangle{\bullet}g\} \text{ (where } g\text{: }\mathbf{rgn}) & \mathrm{ftpt}(G_1{\bullet}f \subseteq G_2) & = \mathrm{ftpt}(G_1) \cup \mathrm{ftpt}(G_2) \cup \{\mathbf{rd}\ G_1{\bullet}f\}
\end{array}$$

Fig. 5 defines the *frames* judgement $P \vdash \overline{\varepsilon}\ \mathbf{frm}\ P'$. For a primitive, $\mathrm{ftpt}(P)$ is precisely what is needed to evaluate the formula. For any formula, the frames judgement gives a conservative approximation of what must be read.

The rule for $\forall$ in Fig. 5 exploits the bounding region to give a footprint that is "local", unless the bound is **alloc**. It is easier to read the following special case that applies when the formula has no expression like $x.g{\bullet}h$ involving a "pivot" field g: $\mathbf{rgn}$.

$$\frac{\mathrm{ftpt}(G) \subseteq \overline{\varepsilon} \qquad P \wedge x \in G \vdash \overline{\varepsilon}, \mathbf{rd}\,x, \mathbf{rd}\langle x \rangle{\bullet}\overline{f}\ \ \mathbf{frm}\ \ Q}{P \vdash \overline{\varepsilon}, \mathbf{rd}\ G{\bullet}\overline{f}\ \mathbf{frm}\ \forall x\text{: } K \in G \mid Q}$$

The $\forall$ rule in Fig. 5 is for the special case of exactly one pivot $x.g$. It generalizes easily to multiple pivot expressions.

$$\frac{P \text{ is primitive}}{\mathbf{true} \vdash \mathrm{ftpt}(P)\ \mathbf{frm}\ P} \qquad \frac{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ Q \qquad Q \Leftrightarrow R}{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ R} \qquad \frac{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ Q}{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ \neg Q}$$

$$\frac{P \vdash \overline{\varepsilon}, \mathbf{rd}\, x\ \mathbf{frm}\ Q}{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ \forall x\colon \mathbf{int} \mid Q} \qquad \frac{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ R \qquad P \wedge R \vdash \overline{\varepsilon}\ \mathbf{frm}\ Q}{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ R \wedge Q}$$

$$\frac{P \vdash \overline{\varepsilon}\ \mathbf{frm}\ R \qquad P \vdash \overline{\varepsilon} \leq \overline{\eta} \qquad Q \Rightarrow P}{Q \vdash \overline{\eta}\ \mathbf{frm}\ R}$$

$$\frac{\begin{array}{c}\mathrm{ftpt}(G) \subseteq \overline{\varepsilon} \\ \mathrm{ftpt}(G') \subseteq \overline{\varepsilon} \qquad P \Rightarrow (\forall x\colon K \in G \mid x.g \subseteq G') \qquad P \wedge x \in G \vdash \overline{\varepsilon}, \mathbf{rd}\, x, \mathbf{rd}\langle x\rangle{\bullet}\overline{f}, \mathbf{rd}\, x.g{\bullet}\overline{h}\ \mathbf{frm}\ Q\end{array}}{P \vdash \overline{\varepsilon}, \mathbf{rd}\, G{\bullet}\overline{f}, \mathbf{rd}\, G'{\bullet}\overline{h}\ \mathbf{frm}\ (\forall x\colon K \in G \mid Q)}$$

Figure 5: Selected rules for frames judgement.

Separators. Given effect sets $\overline{\varepsilon}$ and $\overline{\eta}$, the *separator* formula $\overline{\varepsilon} \star \overline{\eta}$ is a conjunction of certain disjointnesses. In a state where $\overline{\varepsilon} \star \overline{\eta}$ holds, nothing that is allowed by $\overline{\varepsilon}$ to be read can be written according to $\overline{\eta}$. We define $\overline{\varepsilon} \star \overline{\eta}$ by recursion on effect sets: It distributes conjunctively, by $(\varepsilon, \overline{\varepsilon}) \star \overline{\varepsilon}_1 = (\varepsilon \star \overline{\varepsilon}_1) \wedge (\overline{\varepsilon} \star \overline{\varepsilon}_1)$ and $\varepsilon \star (\varepsilon', \overline{\varepsilon}) = (\varepsilon \star \varepsilon') \wedge (\varepsilon \star \overline{\varepsilon})$. For single effects the definition is:

$$\begin{array}{lcll} \mathbf{rd}\, y \star \mathbf{wr}\, x & = & \text{if } x \equiv y \text{ then } \mathbf{false} \text{ else } \mathbf{true} \\ \mathbf{rd}\, G_1{\bullet}f \star \mathbf{wr}\, G_2{\bullet}g & = & \text{if } f \equiv g \text{ then } G_1 \,\#\, G_2 \text{ else } \mathbf{true} \\ \mathbf{rd}\, \mathbf{alloc} \star \mathbf{wr}\, \mathbf{alloc} & = & \mathbf{false} \\ \varepsilon \star \varepsilon' & = & \mathbf{true} \quad \text{otherwise} \end{array}$$

Note that write effects in $\overline{\varepsilon}$, reads in $\overline{\eta}$, and freshnesses in both, are not relevant.

Aside 4.1 (On quantified regions) Given frames $\overline{\varepsilon}_P$ for P and $\overline{\varepsilon}_Q$ for Q, we can approximate[5] $P * Q$ as

$$P \wedge Q \wedge (\overline{\varepsilon}_P \star \overline{\delta})$$

where $\overline{\delta}$ is obtained from $\overline{\varepsilon}_Q$ by turning **rd** into **wr**. In a given state, the footprints $\overline{\varepsilon}_P$ and $\overline{\varepsilon}_Q$ have fixed interpretations, whereas $P * Q$ allows that there may be several ways of partitioning the heap in support of P and Q. Our logic's use of explicit footprints and ghost variables can be seen as skolemizing the existential implicit in $*$, since $\overline{\varepsilon}_P \star \overline{\delta}$ typically refers to regions involving ghost variables assigned in the (instrumented) program.

To explain this further, let r be a region variable and K be a class with integer field n. Define P and Q as follows:

$$\begin{array}{rcl} P(r) & \iff & \forall x\colon K \in r \mid x.n \leq 0 \\ Q(r) & \iff & \forall x\colon K \in r \mid x.n \geq 0 \end{array}$$

Suppose we extend the assertion language with quantification over region variables. The following formula exhibits the sort of nondeterminacy that is built into the separating conjunction:

[5] In separation logic, $P * Q$ allows the two formulas to refer to some shared variables. We could approximate that by a refinement of $\overline{\varepsilon}_P \star \overline{\delta}$ that only generates region disjointnesses.

$PQ \;\widehat{=}\; \exists r \mid P(r) \wedge Q(\mathbf{alloc}{-}r)$

Our logic does not include quantification for region variables.[6] Thus we can write, e.g., $P(r) \wedge Q(\mathbf{alloc}{-}r)$ as a postcondition but not PQ. Consider the command

$C0 \;\widehat{=}\; x{:=}\,\mathbf{new}\,K; x.n{:=}\,0$

Then we have

$$\{P(r) \wedge Q(\mathbf{alloc}{-}r)\}\; C0\; \{P(r) \wedge Q(\mathbf{alloc}{-}r)\}[\mathbf{wr}\,x, \mathbf{wr}\,r]$$

(adding r by sub-effecting). But there is another instrumentation that satisfies the same specification, namely let $C0' \equiv C0; r{:=}\; r \cup \langle x \rangle$. In some sense, these correspond to different ways of witnessing the existential in PQ. Were we to include quantification over region variables, then both $C0$ and $C0'$ could be used, with the auxiliary elimination rule, to obtain $\{PQ\}\; x{:=}\,\mathbf{new}\,K; x.n{:=}\,0\; \{PQ\}[\mathbf{wr}\,x]$. □

Soundness of rule FRAME hinges on two semantic properties. First, if $\overline{\delta}$ frames P and states σ, σ' agree on $\overline{\delta}$ then they agree on the truth value of P. Second, if transition from σ to σ' is allowed by $\overline{\varepsilon}$ and σ satisfies $\overline{\delta} \star \overline{\varepsilon}$ then σ, σ' agree on $\overline{\delta}$. Here are the formal results in [BNR08].

Lemma 4.2 (agreement) For any σ, σ', any predicates P, P', and any set of effects $\overline{\varepsilon}$, suppose $P \vdash \overline{\varepsilon}\ \mathbf{frm}\ P'$, $\sigma \models P$, and σ, σ' agree on $\overline{\varepsilon}$. Then $\sigma \models P'$ iff $\sigma' \models P'$.

For any states, σ, σ', for any expression F, suppose that σ, σ' agree on $\mathrm{ftpt}(F)$. Then $[\![F]\!]\sigma = [\![F]\!]\sigma'$.

Consider any effect sets $\overline{\varepsilon}_1$ and $\overline{\varepsilon}_2$. Suppose $\sigma \rightarrow \sigma' \models \overline{\varepsilon}_2$ and $\sigma \models \overline{\varepsilon}_1 \star \overline{\varepsilon}_2$. Then σ, σ' agree on $\overline{\varepsilon}_1$.

Lemma 4.3 (write sub-effect) If $P \vdash \overline{\varepsilon}_1 \leq \overline{\varepsilon}_2$ and $\sigma \rightarrow \sigma' \models \overline{\varepsilon}_1$ and $\sigma \models P$ then $\sigma \rightarrow \sigma' \models \overline{\varepsilon}_2$.

Lemma 4.4 (read sub-effect) Suppose $P \vdash \overline{\varepsilon}_1 \leq \overline{\varepsilon}_2$ and σ, σ' agree on $\overline{\varepsilon}_2$. If $\sigma \models P$ then σ, σ' agree on $\overline{\varepsilon}_1$.

5 The core program logic, RL

The RL judgement for correctness takes the form

$$\Delta \vdash^{\Gamma} \{P\}\; C\; \{P'\}[\overline{\varepsilon}] \tag{2}$$

where Δ is a set of *method specifications*, each of the form

$$(\overline{y}{:}\,\overline{U})\{Q\}m(\overline{x}{:}\,\overline{T})\{Q'\}[\overline{\eta}] \tag{3}$$

[6] In this it differs from the logics developed by de Boer and Pierik, which have finite sequences.

$$\textsc{Call}\ \frac{\Delta \text{ contains } (\overline{y}\colon \overline{U})\{P\}m(\overline{x}\colon \overline{T})\{P'\}[\overline{\varepsilon}] \qquad Q \equiv P/\overline{x}{\to}\overline{z} \qquad Q' \equiv P'/\overline{x}{\to}\overline{z} \qquad \overline{\varepsilon}' \equiv \overline{\varepsilon}/\overline{x}{\to}\overline{z}}{\Delta \vdash \{Q\}\ m(\overline{z})\ \{Q'\}[\overline{\varepsilon}']}$$

$$\textsc{Var}\quad \frac{\Delta \vdash^{\Gamma, x\colon T} \{P\}\ C\ \{P'\}[\mathbf{wr}\, x, \overline{\varepsilon}]}{\Delta \vdash^{\Gamma} \{P\}\ \mathbf{var}\ x\colon T\ \mathbf{in}\ C\ \mathbf{end}\ \{P'\}[\overline{\varepsilon}]}$$

$$\textsc{Alloc}\quad \{\mathbf{true}\}\ x := \mathbf{new}\ K\ \{x\ \mathbf{is}\ K\}[\mathbf{wr}\, x, \mathbf{wr}\, \mathbf{alloc}, \mathbf{fr}\langle x\rangle]$$

$$\textsc{Seq}\ \frac{\begin{array}{c}\{P\}\ C_1\ \{Q\}[\overline{\varepsilon}_1, \mathbf{fr}\ G] \qquad \{Q\}\ C_2\ \{P'\}[\overline{\varepsilon}_2, \mathbf{wr}\ \overline{G}{\bullet}\overline{f}] \qquad \overline{\varepsilon}_1 \text{ is } \mathbf{fr}\text{-free} \\ Q \Rightarrow G_i \subseteq G \text{ for each } \mathbf{wr}\ \overline{G}_i{\bullet}\overline{f}_i \qquad \overline{\varepsilon}_2 \text{ is } P/\overline{\varepsilon}_1\text{-immune} \qquad G \text{ is } Q/(\overline{\varepsilon}_2, \mathbf{wr}\ \overline{G}{\bullet}\overline{f})\text{-immune}\end{array}}{\{P\}\ C_1\ ;\ C_2\ \{P'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2, \mathbf{fr}\ G]}$$

$$\textsc{While}\quad \frac{R \equiv \bigwedge\{G = \mathbf{emp} \mid \mathbf{fr}\ G \text{ is in } \overline{\varepsilon}\} \qquad \Delta, \{P\}\ m\ \{P \wedge x \neq 0\}[\overline{\varepsilon}] \vdash \{P\}\ m; C\ \{P\}[\overline{\varepsilon}]}{\Delta \vdash \{P \wedge R\}\ \mathbf{while}\ x\ \mathbf{do}\ C\ \{P \wedge x = 0\}[\overline{\varepsilon}]}$$

$$\textsc{Bind}\quad \frac{\Delta, \Delta' \vdash^{\Gamma} \{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad \begin{array}{l}\text{Each } (\overline{y}_i\colon \overline{U}_i)\{Q_i\}m_i(\overline{x}_i\colon \overline{T}_i)\{Q_i'\}[\overline{\varepsilon}_i] \text{ in } \Delta' \\ \text{has } \Delta, \Delta' \vdash^{\Gamma, \overline{x}_i\colon \overline{T}_i, \overline{y}_i\colon \overline{U}_i} \{Q_i\}\ \overline{B}_i\ \{Q_i'\}[\overline{\varepsilon}_i]\end{array}}{\Delta \vdash^{\Gamma} \{P\}\ \mathbf{let}\ \mathit{sigs}(\Delta')\ \mathbf{be}\ \overline{B}\ \mathbf{in}\ C\ \{P'\}[\overline{\varepsilon}]}$$

$$\textsc{Assign}\quad \frac{y \not\equiv x}{\{x = y\}\ x := F\ \{x = (F/x{\to}y)\}[\mathbf{wr}\, x]}$$

$$\textsc{FldAcc}\ \frac{z \not\equiv x}{\{y \neq \mathbf{null} \wedge z = y\}\ x := y.f\ \{x = z.f\}[\mathbf{wr}\, x]}$$

$$\textsc{If}\quad \frac{\{P \wedge x \neq 0\}\ C_1\ \{P'\}[\overline{\varepsilon}] \qquad \{P \wedge x = 0\}\ C_2\ \{P'\}[\overline{\varepsilon}]}{\{P\}\ \mathbf{if}\ x\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{P'\}[\overline{\varepsilon}]}$$

Figure 6: Syntax directed rules of RL.

where m is a method name, $\overline{x}$ are the parameter names and $\overline{y}$ the names of auxiliary variables used in the pre/post conditions Q, Q' and in the effect $\overline{\eta}$. For (3) to be well formed, $Q, Q', \overline{\eta}$ should be wf in $\Gamma, \overline{y}\colon \overline{U}, \overline{x}\colon \overline{T}$. Less obviously, $\overline{\eta}$ must not contain $\mathbf{wr}\, z$ for any z in $\overline{y}, \overline{x}$; this enforces the usual constraint on value-parameters, so their use in postconditions refers to their initial value. Judgement (2) is well formed just if P, P', $\overline{\varepsilon}$, and Δ are well formed in Γ, and moreover $\Gamma; \mathit{sigs}(\Delta) \vdash C$ where *sigs* extracts the method signatures from a set of method specifications.

Selected proof rules are in Figs. 6 and Fig. 7. For readability, we omit the type assignment to variables (Γ) in each rule where it is the same (and unconstrained) in all the antecedent and consequent correctness judgements. We also omit "$\Delta \vdash$" in most cases where the antecedents and consequents use the same Δ and impose no constraints on it.

N.B. that the rules are not formulated to preserve well-formedness. Rather, they are to be instantiated only with well formed antecedents and consequents. For example, in rule VAR the formulas P, P' cannot mention x since it is not in dom(Γ) as would be

required for the consequent to be well formed. Another example is EXIST, where x cannot occur in P'. In typical use, EXIST is used with P' of the form $\exists x' \mid Q/x{\rightarrow}x'$ for some Q from which P' is obtained by CONSEQ. The renaming is necessary to conform to our hygiene condition.

Sometimes we distinguish between *proper rules* and *axioms* for correctness judgements; axioms have no antecedent correctness judgements (though they may have "side conditions" written above the inference line, e.g., in CALL). Unqualified, "rule" encompasses both.

In a sequence like $y{:=}\,\mathbf{new}\,K; y.f{:=}\,0$, the effect $\mathbf{wr}\langle y\rangle{\bullet}f$ of the update cannot be retained as an effect of the sequence, since in the initial state of the sequence y has a different value than in the initial state of the update. Rule SEQ removes writes to fresh objects and uses immunity conditions to ensure that the remaining effects are sensibly propagated. Region expression G is $P/\overline{\varepsilon}$-*immune* provided $P \Rightarrow \mathrm{ftpt}(G) \star \overline{\varepsilon}$ is valid. Effect set $\overline{\varepsilon}_2$ is $P/\overline{\varepsilon}_1$-*immune* provided that for all G, f such that $\mathbf{wr}\,G{\bullet}f$ occurs in $\overline{\varepsilon}_2$, G is $P/\overline{\varepsilon}_1$-immune.

Rule WHILE expresses the induction step in relational form, using an assumption that stands for the previous iterations. This form is needed to deal with writes to objects that were allocated in previous iterations.[7]

Programming rule SUBEFF, provides for use of sub-effect relations, which are interpreted in the pre-state. Rule SUBFR essentially provides for sub-effecting of freshness effects, which are interpreted in the post-state. There are other sound rules that connect pre- and post-conditions with effects. For example, a temporary update can be deleted according to this rule:

$$\textsc{NoUpd}\quad \frac{\{P\}\ C\ \{P'\}[\mathbf{wr}\langle x\rangle{\bullet}f, \overline{\varepsilon}] \qquad \mathbf{rd}\,x \star \overline{\varepsilon} \qquad \mathbf{rd}\,y \star \overline{\varepsilon} \qquad P \vee P' \Rightarrow x.f = y}{\{P\}\ C\ \{P'\}[\overline{\varepsilon}]}$$

Freshness can be expressed in the postcondition by the rule

$$\textsc{FrPost}\quad \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}, \mathbf{fr}\ G] \qquad P \Rightarrow \mathrm{ftpt}(G') \star \overline{\varepsilon}}{\{P\}\ C\ \{P' \wedge G\ \#\ G'\}[\overline{\varepsilon}, \mathbf{fr}\ G]}$$

The side condition in FRPOST ensures that the final value of G' is the same as its initial value, and thus it contains only preexisting objects.[8]

Rule AUX is for elimination of local ghost variables (auxiliaries, in the sense of Owicki and Gries) within a method body. The side condition checks that the variables $\overline{v}$ only occur in assignments to variables in $\overline{v}$; function *erase* replaces those assignments by **skip**. Method specifications also involve auxiliary variables, which cannot be directly eliminated using AUX; they can be eliminated using SUBST and CONSEQ. We do not formalize the elimination of ghost fields.

[7] Within the loop body, SEQ is used together with effect subsumption and local ghost variables that snapshot regions to obtain effects with suitable immunity. SEQ loses no generality by insisting that C_1 establish a freshness and C_2 have field writes: by rules for the sub-effect judgement, $\mathbf{fr}\,\mathbf{emp}$ is equivalent to the empty effect set, as is $\mathbf{wr}\,\mathbf{emp}{\bullet}f$.

[8] This style could perhaps replace the **fr** effect and avoid the need for an unusual While rule.

$$\textsc{Frame}\ \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad P \vdash \overline{\delta}\ \mathbf{frm}\ Q \qquad P \Rightarrow \overline{\delta} \star \overline{\varepsilon}}{\{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}]}$$

$$\textsc{Assum}\ \frac{\Delta \vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}]}{\Delta, \Delta' \vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}]} \qquad \textsc{SubEff}\ \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad P \vdash \overline{\varepsilon} \leq \overline{\varepsilon}'}{\{P\}\ C\ \{P'\}[\overline{\varepsilon}']}$$

$$\textsc{SubFr}\ \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}, \mathbf{fr}\ G] \qquad P' \vdash G' \subseteq G}{\{P\}\ C\ \{P'\}[\overline{\varepsilon}, \mathbf{fr}\ G']}$$

$$\textsc{Subst}\ \frac{\{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad (P/x{\rightarrow}F) \Rightarrow \mathrm{ftpt}(F) \star (\overline{\varepsilon}/x{\rightarrow}F) \qquad \mathbf{rd}\ x \notin \overline{\varepsilon} \qquad \mathbf{wr}\ x \notin \overline{\varepsilon}}{\{P/x{\rightarrow}F\}\ C\ \{P'/x{\rightarrow}F\}[\overline{\varepsilon}/x{\rightarrow}F]}$$

$$\textsc{Conj}\quad \frac{\{P_1\}\ C\ \{P_1'\}[\overline{\varepsilon}] \qquad \{P_2\}\ C\ \{P_2'\}[\overline{\varepsilon}]}{\{P_1 \wedge P_2\}\ C\ \{P_1' \wedge P_2'\}[\overline{\varepsilon}]} \qquad \textsc{Exist}\quad \frac{\vdash^{\Gamma, x:T} \{P\}\ C\ \{P'\}[\overline{\varepsilon}]}{\vdash^{\Gamma} \{\exists x\colon T \in G \mid P\}\ C\ \{P'\}[\overline{\varepsilon}]}$$

$$\textsc{Conseq}\quad \frac{\{P_1\}\ C\ \{P_1'\}[\overline{\varepsilon}] \qquad P_2 \Rightarrow P_1 \qquad P_1' \Rightarrow P_2'}{\{P_2\}\ C\ \{P_2'\}[\overline{\varepsilon}]}$$

$$\textsc{UniFr}\ \frac{\vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}, \mathbf{fr}\ G, \mathbf{fr}\ G']}{\vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}, \mathbf{fr}\ G \cup G']} \qquad \textsc{Context}\ \frac{\vdash^{\Gamma} \{P\}\ C\ \{P'\}[\overline{\varepsilon}]}{\vdash^{\Gamma, x:T} \{P\}\ C\ \{P'\}[\overline{\varepsilon}]}$$

$$\textsc{Aux}\ \frac{\{P\}\ \mathbf{var}\ \overline{v}\colon \overline{T}\ \mathbf{in}\ C\ \mathbf{end}\ \{P'\}[\overline{\varepsilon}] \qquad \mathit{auxil}(\overline{v}, C)}{\{P\}\ \mathit{erase}(\overline{v}, C)\ \{P'\}[\overline{\varepsilon}]}$$

Figure 7: Structural rules of RL; $P/x{\rightarrow}F$ denotes capture-avoiding substitution.

Semantics. First we define what it means for a state transformer to satisfy a specification. Then we define satisfaction for method environments, using a suitable lifting operator to deal with auxiliaries in method specifications.

Let t be a state transformer for Γ. Define $t \models^{\Gamma} \{P\} - \{P'\}[\overline{\varepsilon}]$ iff for all Γ-states σ, σ' such that $\sigma \models P$ we have $t(\sigma) \neq \Uparrow$ and if $t(\sigma) = \sigma'$ then $\sigma' \models P'$ and $\sigma \rightarrow \sigma' \models \overline{\varepsilon}$.

If $\Gamma, \overline{y}\colon \overline{U}$ is well formed and t is a state transformer for Γ, define $t \otimes \overline{y}\colon \overline{U}$ to be the state transformer for $\Gamma, \overline{y}\colon \overline{U}$ that discards $\overline{y}$ from the initial state and applies t; and if t returns a state, it gets extended with $\overline{y}$ mapped to their initial values; otherwise, the outcome ($\bot$ or $\Uparrow$) is retained. Let μ be a method environment for $\Gamma; \mathit{sigs}(\Delta)$. Define $\mu \models^{\Gamma} \Delta$ iff $\mu(m) \otimes \overline{y}\colon \overline{U} \models^{\Gamma, \overline{x}:\overline{T}, \overline{y}:\overline{U}} \{P\} - \{P'\}[\overline{\varepsilon}]$ for every specification $(\overline{y}\colon \overline{U})\{P\}m(\overline{x}\colon \overline{T})\{P'\}[\overline{\varepsilon}]$ in Δ. Judgement $\Delta \vdash^{\Gamma} \{P\}\ C\ \{P'\}[\overline{\varepsilon}]$ is *valid* iff for all μ such that $\mu \models^{\Gamma} \Delta$ we have $[\![\Gamma; \mathit{sigs}(\Delta) \vdash C]\!]\mu \models^{\Gamma} \{P\} - \{P'\}[\overline{\varepsilon}]$.

A judgement is *derivable* if it can be obtained using well formed rule instantiations where the side conditions hold (i.e., formulas are valid and the framing and sub-effecting judgements are derivable).

The main result of [BNR08] is that every rule preserves validity, whence the following.

Proposition 5.1 (RL is sound) If $\Delta \vdash^{\Gamma} \{P\}\ C\ \{P'\}[\overline{\varepsilon}]$ is derivable then it is valid. □

Soundness of SEQ is shown using the following.

Lemma 5.2 (expression immunity) Let G be $P/\overline{\varepsilon}$-immune. Then $[\![G]\!]\sigma = [\![G]\!]\sigma'$ for any σ, σ' such that $\sigma \to \sigma' \models \overline{\varepsilon}$ and $\sigma \models P$.

Lemma 5.3 (command immunity) Suppose $\overline{\varepsilon}_1$ is **fr**-free, $\overline{\varepsilon}_2$ is $P/\overline{\varepsilon}_1$-immune, G is $Q/(\overline{\varepsilon}_2, \mathbf{wr}\, \overline{G}\bullet\overline{f})$-immune, and $Q \Rightarrow G_i \subseteq G$ for each $\mathbf{wr}\, \overline{G}_i\bullet\overline{f}_i$. Suppose $\sigma \to \sigma_1 \models \overline{\varepsilon}_1, \mathbf{fr}\, G$ and $\sigma_1 \to \sigma' \models \overline{\varepsilon}_2, \mathbf{wr}\, \overline{G}\bullet\overline{f}$ and $\sigma \models P$ and $\sigma_1 \models Q$. Then $\sigma \to \sigma' \models \overline{\varepsilon}_1, \overline{\varepsilon}_2, \mathbf{fr}\, G$.

6 On second order framing

In [BNR08], the following example is used to motivate the second order frame rule, which is introduced there informally.

The following classes can be used for a set represented by a linked list that may contain duplicate elements.

$$\begin{array}{l}\mathbf{class}\ \mathit{Coll}\ \{\mathit{rep}\colon \mathbf{rgn};\ \mathit{lst}\colon \mathit{Node};\ \mathit{size}\colon \mathbf{int};\}\\ \mathbf{class}\ \mathit{Node}\ \{\mathit{item}\colon T;\ \mathit{len}\colon \mathbf{int};\ \mathit{nxt}\colon \mathit{Node};\}\end{array}$$

Here is an illustrative object invariant, using sugared syntax:

$$\begin{array}{rcl}\mathit{CollI}(c) & \widehat{=} & c\,\mathbf{is}\,\mathit{Coll} \wedge c.\mathit{lst} \in c.\mathit{rep} \wedge \\ & & c.\mathit{rep}\bullet\mathit{nxt} \subseteq c.\mathit{rep} \wedge c.\mathit{lst}.\mathit{len} \geq c.\mathit{size}\end{array}$$

The following can be derived:

$$\mathbf{true} \vdash \overline{\varepsilon}_{\mathit{CollI}}\ \mathbf{frm}\ \mathit{CollI}(c) \tag{4}$$

where $\overline{\varepsilon}_{\mathit{CollI}} \equiv \mathbf{rd}\, c, \langle c\rangle\bullet(\mathit{rep}, \mathit{lst}, \mathit{size}), c.\mathit{rep}\bullet(\mathit{len}, \mathit{nxt})$.

One might expect to verify the body, B, of a method specified as $\{P\}\, m(d)\, \{P'\}[\overline{\varepsilon}]$ by proving $\{P \wedge \mathit{CollI}(d)\}\, B\, \{P' \wedge \mathit{CollI}(d)\}[\overline{\varepsilon}]$. Instead, we suggest that B be verified using a module invariant that pertains to all instances:

$$Q_0 \ \widehat{=}\ \ \forall c\colon \mathit{Coll} \in \mathit{cs} \mid \mathit{CollI}(c)$$

This does not abandon local reasoning, as we shall explain. First, to frame the formula Q_0, suppose global variable cr holds the union of the reps of cs, i.e.

$$\mathit{CollC}_0 \ \widehat{=}\ \ \forall c \in \mathit{cs} \mid c.\mathit{rep} \subseteq \mathit{cr}$$

Formula CollC_0 says that the module's internal representation objects are in cr. This serves to derive, from (4), the framing

$$\mathit{CollC}_0 \vdash \overline{\delta}_0\ \mathbf{frm}\ Q_0$$

where $\overline{\delta}_0 \equiv \mathbf{rd}\, \mathit{cs}, \mathit{cr}, \mathit{cs}\bullet(\mathit{rep}, \mathit{lst}, \mathit{size}), \mathit{cr}\bullet(\mathit{len}, \mathit{nxt})$. We can express a "package confinement" [GPV06] condition, that clients don't reach reps, as follows:[9]

[9] Here "any" stands for all field names in the program.

$$CollC_1 \;\widehat{=}\; (\mathbf{alloc}{-}(cs{\cup}cr))_{\bullet\mathsf{any}} \# cr$$

This could well be an all-states invariant enforced by a type system [GPV06]. Now, this example illustrates owner-style encapsulation, where each collection d owns its own "reps", and $CollI(d)$ depends only on those. This separation is captured by

$$CollC_2 \;\widehat{=}\; \forall c, c'{:}\, Coll \in cs \mid c = c' \vee c.rep \# c'.rep$$

Suppose the implementation B only acts on d and its reps. Then, using $CollC_2$ etc, the footprint of B is disjoint from the footprint of the formula

$$\forall c \in cs{-}\langle d\rangle \mid CollI(c) \tag{5}$$

Hence, from $\{P \wedge CollI(d)\}\ B\ \{P' \wedge CollI(d)\}[\overline{\varepsilon}]$ we can derive $\{P \wedge Q_0\}\ B\ \{P' \wedge Q_0\}[\overline{\varepsilon}]$ using FRAME for (5) and then consequence.

Having argued for Q_0 as suitable module invariant, we exhibit Hoare's mismatch as an instance of the second order frame rule. First, define the *confinement invariant* $R_0 \;\widehat{=}\; CollC_0 \wedge CollC_1 \wedge CollC_2$ and the *client effect bound*

$$\overline{\eta}_0 \;\widehat{=}\; \mathbf{wr}(\mathbf{alloc}{-}(cs{\cup}cr))_{\bullet\mathsf{any}}, \mathbf{wr}(\mathrm{dom}(\Gamma){-}\{cs, cr\})$$

that disallows client writes of the variables cs, cr and of objects in those regions. Now we have this instance of FRAME2, eliding effects for now.

$$\frac{\Delta; \Delta^* \vdash_{R_0;\overline{\eta}_0} \{P\}\ C\ \{P'\}[\ldots] \qquad R_0 \vdash \overline{\delta}_0\ \mathbf{frm}\ Q_0 \qquad R_0 \Rightarrow \overline{\delta}_0 \star \overline{\eta}_0}{\Delta, (\Delta^* \otimes Q_0) \vdash \{P \wedge Q_0\}\ C\ \{P' \wedge Q_0\}[\ldots]}$$

The antecedent says some client C is verified with respect to an ambient library, Δ, and some specifications Δ^* treated as a module with hidden invariant Q_0. The consequent says that the implementations are subject to proof obligation $\Delta^* \otimes Q_0$ which means Q_0 is added as explicit pre- and post-condition of each specification in Δ^*. Once those assumptions are discharged, what has been verified about C is $\{P \wedge Q_0\}\ C\ \{P' \wedge Q_0\}[\ldots]$, which is enough because in a complete program Q_0 is established by module initialization.

The antecedent correctness judgement of FRAME2 is in an extended logic, EL, that propagates R_0 and $\overline{\eta}_0$ through the subproof. The EL judgements also separate the effects of "client code", which must be bounded by $\overline{\eta}_0$, from those of the module procedures which are likely to write state read by Q_0. Condition $R_0 \Rightarrow \overline{\delta}_0 \star \overline{\eta}_0$ says that, in R_0-states, effects within the client effect bound $\overline{\eta}_0$ do not interfere with the frame of Q_0, which is given by $R_0 \vdash \overline{\delta}_0\ \mathbf{frm}\ Q_0$. These two side conditions in the rule are what is necessary for the discipline embodied by $R_0, \overline{\eta}_0$ to be sound.

Current verifiers typically impose a single discipline on all programs, so one can imagine that the proof for each method body uses a single instance of FRAME2, as its last step. But the logic EL allows multiple uses of FRAME2, so that different sub-proofs can involve different disciplines.

A type-based [MPHL06] or assertion-based [LM04, NB04] discipline will impose the confinement invariant and client effect bound at every program point. This is not necessary: Both can be violated within large subprograms that do not have calls to methods in

Δ^*. Such subprograms are verified within the RL sub-logic of EL, via the rule RLtoEL given in Sec. 7.

Generalizing from the example, note that we need one or more handles, like c in *CollI*, on the cluster of objects that is considered to be one instance of the module. But the level of granularity can be coarser than ownership, e.g., a collection together with multiple (client-accessible) iterators and their shared representation.

7 The extended logic, EL

EL judgements. The context is partitioned into the form $\Delta; \Delta^*$, where Δ and Δ^* are both sets of method specifications, with no method name in common.[10] Those in Δ are treated just as in RL; one may think of it as the interface for the ambient libraries. Those in Δ^* are associated with a designated module and are the focus of the higher order frame rule. The effect clause is partitioned[11] into the form $\overline{\varepsilon}; \overline{\varepsilon}^*$. Effects for calls to procedures in Δ^* are put in $\overline{\varepsilon}^*$, all other effects into $\overline{\varepsilon}$. Finally, the turnstile is subscripted with a *confinement invariant* R and *client effect bound* $\overline{\eta}$. So an *EL judgement* takes the form $\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]$. It is well formed just if $R, P, P', \overline{\eta}, \overline{\varepsilon}, \overline{\varepsilon}^*$ are well formed in Γ.

Both R and $\overline{\eta}$ are propagated unchanged through the rules for this judgement, and are not otherwise mentioned in those rules —with the exception of Frame2 and also RLtoEL which is given below.

Validity of EL judgements is defined in terms of a corresponding RL judgement that forgets R and $\overline{\eta}$. This is used as a sanity check on the EL rules but as noted earlier the semantics does not account for the key rule, Frame2.

Definition 7.1 (valid EL judgement) An EL judgment $\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]$ is *valid* iff the associated RL judgement $\Delta, \Delta^* \vdash^{\Gamma} \{P\}\ C\ \{P'\}[\overline{\varepsilon}, \overline{\varepsilon}^*]$ is.[12]

EL rules. For each of the proper rules of RL except Bind —but not the axioms— there is a corresponding EL rule; see Fig. 8. Rules If*, Seq*, and While* resemble Var*. The omitted structural rules are also straightforward (see appendix). There are two additional EL rules. The first converts an RL judgement to an EL judgement with arbitrarily chosen Δ^* but imposing the constraints[13] R and $\overline{\eta}$:

$$\text{RLtoEL}\ \frac{\Delta \vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}] \qquad P \Rightarrow R \qquad P \vdash \overline{\varepsilon} \leq \overline{\eta}}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \varnothing]}$$

[10] The symbol ; binds less tightly than comma. The asterisk in "Δ^*" is not a mathematical operator but merely a way to decorate the letter to obtain a fresh identifier.

[11] It would be sound, and simpler, to have a single effect set. However, we want to allow the focused module's methods to act on the encapsulated data structures, and the invariant to involve that, while insisting effects by "client" method calls and primitives (those in the "RLtoEL subtrees") do not interfere. So the partitioning is needed for completeness.

[12] Here and elsewhere we refrain from mentioning that the judgements involved should be well formed.

[13] Technically, it is possible to dispense with $R, \overline{\eta}$ and instead thread the invariant's footprint, $\overline{\delta}$, through the EL judgements, using in RLtoEL the condition $P \Rightarrow \overline{\delta} \star \overline{\varepsilon}$. But this loses the explicit separation discipline.

$$\textsc{Var}^* \ \frac{\Delta; \Delta^* \vdash^{\Gamma, x\colon T}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\mathbf{wr}\, x, \overline{\varepsilon}; \overline{\varepsilon}^*]}{\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ \mathbf{var}\ x\colon T\ \mathbf{in}\ C\ \mathbf{end}\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Call}^* \ \frac{\begin{array}{c}\Delta^* \text{ contains } (\overline{y}\colon \overline{U})\{P\}m(\overline{x}\colon \overline{T})\{P'\}[\overline{\varepsilon}] \\ Q \equiv P/\overline{x}{\to}\overline{z} \qquad Q' \equiv P/\overline{x}{\to}\overline{z} \qquad \overline{\varepsilon}' \equiv \overline{\varepsilon}/\overline{x}{\to}\overline{z}\end{array}}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{Q\}\ m(\overline{z})\ \{Q'\}[\varnothing; \overline{\varepsilon}']}$$

$$\textsc{Frame}^* \ \frac{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad P \vdash \overline{\delta}\ \mathbf{frm}\ Q \qquad P \Rightarrow \overline{\delta} \star (\overline{\varepsilon}, \overline{\varepsilon}^*)}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Exist}^* \ \frac{\vdash^{\Gamma, x\colon T}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}{\vdash^{\Gamma}_{R;\overline{\eta}} \{\exists x\colon T \in G \mid P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Subst}^* \ \frac{\begin{array}{c}\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \\ (P/x{\to}F) \Rightarrow \mathrm{ftpt}(F) \star ((\overline{\varepsilon}, \overline{\varepsilon}^*)/x{\to}F) \qquad \mathbf{rd}\, x \notin \overline{\varepsilon}, \overline{\varepsilon}^* \qquad \mathbf{wr}\, x \notin \overline{\eta}, \overline{\varepsilon}, \overline{\varepsilon}^*\end{array}}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P/x{\to}F\}\ C\ \{P'/x{\to}F\}[\overline{\varepsilon}/x{\to}F; \overline{\varepsilon}^*/x{\to}F]}$$

$$\textsc{Conj}^* \ \frac{\vdash_{R;\overline{\eta}} \{P_1\}\ C\ \{P_1'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad \vdash_{R;\overline{\eta}} \{P_2\}\ C\ \{P_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}{\vdash_{R;\overline{\eta}} \{P_1 \wedge P_2\}\ C\ \{P_1' \wedge P_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

Figure 8: Selected correctness rules for EL.

For example, one can use the RL axiom Call followed by RLtoEL for calls to methods in Δ whereas Call* is only for methods in Δ^*.

The last EL rule is the second order frame rule:

$$\textsc{Frame2} \ \frac{\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad R \vdash \overline{\delta}\ \mathbf{frm}\ Q \qquad R \Rightarrow \overline{\delta} \star \overline{\eta}}{\Delta, (\Delta^* \otimes Q) \vdash^{\Gamma} \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}$$

Whereas the antecedent of RLtoEL is an RL judgement and the consequent an EL judgement, the reverse is true of Frame2. The rule uses a notation from separation type systems [BTSY05] to add an invariant, Q, to the pre- and post-conditions of some specifications. Let Δ and Q be well formed in Γ. Define $\Delta \otimes Q$ to be the set of method specifications $(\overline{y}\colon \overline{U})\{P \wedge Q\}m(\overline{x}\colon \overline{T})\{P' \wedge Q\}[\overline{\varepsilon}]$ such that Δ contains $(\overline{y}\colon \overline{U})\{P\}m(\overline{x}\colon \overline{T})\{P'\}[\overline{\varepsilon}]$.

The extended logic, EL. The extended logic, EL for short, involves both RL and EL judgements. We say "EL rules" for the rules in Fig. 8 together with the special rules RLtoEL and Frame2 (and those in the appendix Fig. 9). The logic EL consists of the RL rules together with the EL rules, as well as the rules for the subsidiary judgements already present in RL.

For proving RL-judgements, there is no way to use EL-judgements or rules except as needed to use Frame2, since no other EL rule derives an RL-judgement.

In a derivation of an EL-judgement $\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]$, if Frame2 is not used then every use of Call* is instantiated with $R, \overline{\eta}, \Delta^*$ (rather than some other R',

$\overline{\eta}'$, or Δ'^*), because the other EL rules preserve $R, \overline{\eta}, \Delta^*$ unchanged.

The main result of the paper is that FRAME2 is admissible. That implies soundness of EL, given the following.

Lemma 7.2 (EL rule soundness) Aside from FRAME2, every EL rule is sound: if its antecedents (if any) are valid and the side conditions hold then the consequent is valid. □

For the "forgetful" semantics of Def. 7.1, FRAME2 is patently unsound. For example, instantiate FRAME2 with $R \equiv \mathbf{true}$, $Q \equiv (x = 0)$, $C \equiv x := 1$, $P \equiv \mathbf{true}$, $P' \equiv \mathbf{true}$, $\overline{\varepsilon} \equiv \mathbf{wr}\, x$, $\overline{\eta} = \varnothing$, and $\overline{\delta} \equiv \mathbf{rd}\, x$. Then $\overline{\delta} \star \overline{\eta}$ is valid and the rule's side conditions hold. Moreover, under the forgetful semantics the antecedent $\vdash_{R;\overline{\eta}} \{\mathbf{true}\}\, x := 1\, \{\mathbf{true}\}[\mathbf{wr}\, x; \varnothing]$ is valid but not so the consequent $\vdash \{x = 0\}\, x := 1\, \{x = 0\}[\mathbf{wr}\, x]$.

It is straightforward to prove the Lemma (see appendix): Aside from FRAME2, the EL rules merely propogate the confinement invariant and client effect bound and otherwise mimic their RL counterparts.

8 The extended logic is sound

Recall the transformation described in the first paragraph of Sect. 1. By augmenting each use of RLTOEL with a suitable use of FRAME, and conjoining the invariant everywhere except in the subproofs for antecedents of RLTOEL, we can prove:

Theorem 8.1 (admissibility of Frame2) For any RL or EL judgement provable in EL, there is a proof (in EL) in which FRAME2 is not used.

Proof: By induction on the given derivation D of the given judgement J, and by cases on the last rule in D. If the last rule is anything besides FRAME2, the induction hypothesis yields derivations for the antecedents (if any) without using FRAME2; these, followed by the last rule instance, yield a derivation of J without using FRAME2.

If the last rule is FRAME2, the induction hypothesis yields a derivation D' of the antecedent correctness judgement without any use of FRAME2. To conclude the proof in this case, we appeal to Lemma 8.2 for D'. □

Lemma 8.2 (Frame2 elimination) Suppose judgement

$$\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \tag{6}$$

has a derivation D that does not use FRAME2. Suppose that $R \vdash \overline{\delta}\ \mathbf{frm}\ Q$ is derivable, $R \Rightarrow \overline{\delta} \star \overline{\eta}$ is valid, and Q is well formed in Γ, so that FRAME2 can be instantiated with antecedent (6) to yield

$$\Delta, (\Delta^* \otimes Q) \vdash^{\Gamma} \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*] \tag{7}$$

Then there is a derivation of (7) that does not use FRAME2, nor any other EL rule.

Proof: The proof is by induction on D and by cases on the last rule in D.

If the last rule is CALL*, then (7) can be obtained directly because the requisite specification is in $\Delta^* \otimes Q$. In detail, since the rule's consequent is (6) the instance must look like

$$\frac{\Delta^* \text{ contains } (\overline{y}\colon \overline{U})\{P_0\}m(\overline{x}\colon \overline{T})\{P_0'\}[\overline{\varepsilon}_0] \qquad P \equiv P_0/\overline{x}{\to}\overline{z} \qquad P' \equiv P_0/\overline{x}{\to}\overline{z} \qquad \overline{\varepsilon}^* \equiv \overline{\varepsilon}_0/\overline{x}{\to}\overline{z}}{\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ m(\overline{z})\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

(and $\overline{\varepsilon}$ is $\varnothing$). By definition, $\Delta^* \otimes Q$ contains $(\overline{y}\colon \overline{U})\{P_0 \wedge Q\}m(\overline{x}\colon \overline{T})\{P_0' \wedge Q\}[\overline{\varepsilon}_0]$, and thus the union $\Delta, (\Delta^* \otimes Q)$ also contains this method specification. Since Q and Δ^* are wf in Γ, no variable in $\overline{x}$ is in Q, so $(P_0 \wedge Q)/\overline{x}{\to}\overline{z} \equiv (P_0/\overline{x}{\to}\overline{z}) \wedge Q$. Now we can get (7) for $m(\overline{z})$ by instantiating rule CALL.

If the last rule is RLTOEL, the situation looks like this:

$$\frac{\begin{array}{c}\vdots\\ \Delta \vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}]\end{array} \qquad P \Rightarrow R \qquad R \vdash \overline{\varepsilon} \leq \overline{\eta}}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \varnothing]}\ \text{RLTOEL} \tag{8}$$

(and $\overline{\varepsilon}^*$ in (6) is $\varnothing$). We replace it by an instance of FRAME for Q, followed by ASSUM to add $\Delta^* \otimes Q$, as follows:

$$\frac{\dfrac{\begin{array}{c}\vdots\\ \Delta \vdash \{P\}\ C\ \{P'\}[\overline{\varepsilon}]\end{array} \qquad P \vdash \overline{\delta}\ \mathbf{frm}\ Q \qquad P \Rightarrow \overline{\delta} \star \overline{\varepsilon}}{\Delta \vdash \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}]}\ \text{FRAME}}{\Delta, (\Delta^* \otimes Q) \vdash \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}]}\ \text{ASSUM}$$

The side conditions for FRAME are justified as follows. We have $P \Rightarrow R$ from (8) and $R \vdash \overline{\delta}\ \mathbf{frm}\ Q$ by hypothesis, so we get the first side condition, $P \vdash \overline{\delta}\ \mathbf{frm}\ Q$. We have $R \vdash \overline{\varepsilon} \leq \overline{\eta}$ from (8) and $R \Rightarrow \overline{\delta} \star \overline{\eta}$ by hypothesis so we get $R \Rightarrow \overline{\delta} \star \overline{\varepsilon}$ by the right-monotonicity property of separators (i.e., if $R \Rightarrow \overline{\delta} \star \overline{\eta}$ and $R \vdash \overline{\varepsilon} \leq \overline{\eta}$ then $R \Rightarrow \overline{\delta} \star \overline{\varepsilon}$). Then using $P \Rightarrow R$ again we get the second side condition, $P \Rightarrow \overline{\delta} \star \overline{\varepsilon}$.

If the last rule of D is any other rule, it must be one of the proper rules of EL (like those in Fig. 8). For each of these rules, call it X*, the argument is similar: By induction, we obtain for each of the antecedent judgements a derivation (without FRAME2) of the judgment with Q conjoined to pre- and post-condition; this yields judgements of the form $\Delta, (\Delta^* \otimes Q) \vdash \{P \wedge Q\}\, B\, \{P' \wedge Q\}[\overline{\varepsilon}']$ to which the RL rule X can be applied to obtain (7). In some cases a bit of extra reasoning is needed. For example, consider this derivation ending with an instance of CONJ*:

$$\frac{\begin{array}{c}\vdots\\ \Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P_1\}\ C\ \{P_1'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]\end{array} \qquad \begin{array}{c}\vdots\\ \Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P_2\}\ C\ \{P_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]\end{array}}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P_1 \wedge P_2\}\ C\ \{P_1' \wedge P_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \text{CONJ*}$$

By induction we get the antecedents in the derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta, (\Delta^* \otimes Q) \vdash \{P_1 \wedge Q\}\ C\ \{P_1' \wedge Q\}[\overline{\varepsilon}; \overline{\varepsilon}^*]} \quad \dfrac{\vdots}{\Delta, (\Delta^* \otimes Q) \vdash \{P_2 \wedge Q\}\ C\ \{P_2' \wedge Q\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}}{\Delta, (\Delta^* \otimes Q) \vdash \{P_1 \wedge Q \wedge P_2 \wedge Q\}\ C\ \{P_1' \wedge Q \wedge P_2' \wedge Q\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Conj}}{\Delta, (\Delta^* \otimes Q) \vdash \{P_1 \wedge P_2 \wedge Q\}\ C\ \{P_1' \wedge P_2' \wedge Q\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

where the last step is by Conseq using the tautology $P_1 \wedge Q \wedge P_2 \wedge Q \iff P_1 \wedge P_2 \wedge Q$.

For the case of Frame*, suppose we have

$$\dfrac{\dfrac{\vdots}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]} \qquad P \vdash \overline{\delta}'\ \mathbf{frm}\ Q' \qquad P \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)}{\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P \wedge Q'\}\ C\ \{P' \wedge Q'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Frame}^*$$

We obtain $P \wedge Q \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)$ by logic and $P \wedge Q \vdash \overline{\delta}'\ \mathbf{frm}\ Q'$ by the consequence rule for framing (Fig. 5). By induction we obtain a derivation of the first judgement in

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta, (\Delta^* \otimes Q) \vdash \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]} \quad P \wedge Q \vdash \overline{\delta}'\ \mathbf{frm}\ Q' \quad P \wedge Q \Rightarrow \overline{\delta}' \star (\overline{\varepsilon}, \overline{\varepsilon}^*)}{\Delta, (\Delta^* \otimes Q) \vdash \{P \wedge Q \wedge Q'\}\ C\ \{P' \wedge Q \wedge Q'\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Frame}}{\Delta, (\Delta^* \otimes Q) \vdash \{P \wedge Q' \wedge Q\}\ C\ \{P' \wedge Q' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Conseq}$$

For Exist*, consider a derivation

$$\dfrac{\dfrac{\vdots}{\Delta; \Delta^* \vdash^{\Gamma, x:T}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}}{\Delta; \Delta^* \vdash^{\Gamma}_{R;\overline{\eta}} \{\exists x{:}\ T \in G \mid P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}\ \textsc{Exist}^*$$

By induction we obtain a derivation of the first judgement in

$$\dfrac{\dfrac{\dfrac{\vdots}{\Delta, (\Delta^* \otimes Q) \vdash^{\Gamma, x:T} \{P \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}}{\Delta, (\Delta^* \otimes Q) \vdash^{\Gamma} \{(\exists x{:}\ T \in G \mid P \wedge Q)\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Exist}}{\Delta, (\Delta^* \otimes Q) \vdash^{\Gamma} \{(\exists x{:}\ T \in G \mid P) \wedge Q\}\ C\ \{P' \wedge Q\}[\overline{\varepsilon}, \overline{\varepsilon}^*]}\ \textsc{Conseq}$$

where the Conseq step uses that x does not occur in Q owing to the hypothesis (of the Lemma) that Q is wf in Γ (and the hygiene condition).

For Subst*, we cannot argue that x does not occur in Q as we did in the case of Exist*. For this reason, Subst* augments the side conditions of Subst with side condition $\mathbf{wr}\, x \notin \overline{\eta}$. By hypothesis $R \Rightarrow \overline{\delta} \star \overline{\eta}$ we get that x does not occur in Q, hence $(P \wedge Q)/x{\rightarrow}F \equiv (P/x{\rightarrow}F) \wedge Q$ and the same for P'. □

Corollary 8.3 (conservative extension, soundness of EL) Any RL judgement derivable in EL is derivable in RL. Any judgement derivable in EL is valid.

Proof: By the main Theorem 8.1, any RL judgement derivable in EL can be derived without using rule Frame2. Because no other rule goes from EL judgements to RL judgements, this means it can be derived in RL and is valid by Prop. 5.1.

For any EL judgement derivable in EL, there is a derivation not using Frame2 (Thm. 8.1) and by induction on that derivation we can show the validity of its conclusion, using rule soundness Lemma 7.2 and, for RLtoEL, soundness of RL (Prop. 5.1). □

On precision and a conundrum. The second order frame rule in separation logic (SL) [OYR04] has been proved sound, but only under side conditions on certain of the predicates: they are required to be "precise", meaning roughly that they have unique footprints. Rule Frame2 does not impose such conditions. There are several differences between SL and RL/EL. For lack of a cogent account, we end this section with inconclusive remarks.

Let us write the second order frame rule of SL as follows (slightly adapted from O'Hearn et al. [OYR04]):

$$\frac{\Delta \vdash \{P\}\ C\ \{P'\}}{\Delta * Q \vdash \{P * Q\}\ C\ \{P' * Q\}}$$

In SL, the effect set only tracks writes of variables and we shall ignore it. The assumption set $\Delta * Q$ is obtained by distributing $*Q$ over the pre- and post-conditions in Δ in the same way as our $\Delta \otimes Q$ distributes $\wedge Q$. Reynolds discovered the following problemmatic instance of the rule [OYR04]:

$$\frac{\{0 \vee 1\}\ m\ \{0\} \vdash \{1\}\ m\ \{\mathbf{false}\}}{\{(0 \vee 1) * \mathbf{true}\}\ m\ \{0 * \mathbf{true}\} \vdash \{1 * \mathbf{true}\}\ m\ \{\mathbf{false} * \mathbf{true}\}} \tag{9}$$

Here 0 is short-hand for an SL formula that says the heap is empty, and 1 says the heap contains exactly one cell. The conclusion should surely be invalid: $\mathbf{false} * \mathbf{true} \equiv \mathbf{false}$ yet the assumption $\{(0 \vee 1) * \mathbf{true}\}\, m\, \{0 * \mathbf{true}\}$ is satisfiable, e.g., by skip. The antecedent of the rule is derivable, using the ordinary frame rule of SL together with the conjunction rule. Rather than abandon the conjunction rule, O'Hearn et al. [OYR04] resolve the conundrum by showing that their second order frame rule is sound if (and only if!) the specifications or invariant are restricted to precise predicates. In SL, Q is *precise* if for any heap h there is at most one factorization $h = h' + k$ as a disjoint union of heaps such that h' satisfies Q. The authors show that Q is precise iff $- * Q$ distributes over $\wedge$, i.e., $(P_1 * Q) \wedge (P_2 * Q) \iff (P_1 \wedge P_2) * Q$. Note the resemblance to the tautology that we used in proving admissibility for the case of rule Conj.

Above we should have used the term *pre-heap*, because, in RL, formulas denote sets of complete states, in which heaps are self-contained, whereas SL uses partial heaps in which pointers may be dangling. Indeed, this is critical to SL's ability to delimit command

effects using only pre- and post-conditions. In RL, the truth value of a formula Q may depend only on some fields of some objects —so that its "semantic footprint" is a partial heap— but rather than give semantics for formulas in partial heaps we approximate their footprints using additional notations. Not all RL formulas have unique semantic footprints, e.g., $\exists x{:}\, K \in G \mid x.f = 0$ (to be specific, take $G \equiv \mathbf{alloc}$). On the other hand, if what matters is whether there is a unique splitting of the heap to support P and Q in $P * Q$ then there simply is no counterpart in RL since there is no connective defined in terms of such splittings. (Cf. Aside 4.1). Note that in separation logic, the splitting to support $0 * 1$ is unique, but not so for $(0 \vee 1) * \mathbf{true}$.

Our rule FRAME2 is admissible without imposing a condition like precision on the formulas involved, which poses the question of whether we can derive something like the antecedent $\{0 \vee 1\}\ m\ \{0\} \vdash \{1\}\ m\ \{\mathbf{false}\}$ in (9). Let use use a region variable r intended to capture the footprint of m like the SL specifications do. Define $0 \equiv r = \mathbf{emp}$ and $1 \equiv r = \langle x \rangle$. Reynolds derives $\{0\}\ m\ \{0\}$ from $\{0 \vee 1\}\ m\ \{0\}$ by consequence, whence $\{0 * 1\}\ m\ \{0 * 1\}$ by the ordinary frame rule. Then $\{1\}\ m\ \{1\}$ follows by consequence, as 0 is the unit of $*$. In our setting, approximating $0 * 1$ as suggested in Aside 4.1 would give something of the form $0 \wedge 1 \wedge R$ for some R that implies the footprints of 0 and 1 are disjoint. But $0 \wedge 1 \iff \mathbf{false}$ so we cannot have $1 \Rightarrow 0 \wedge 1 \wedge R$ and the last step cannot go through. Of course one could seek other ways to express $0 * 1$. But in our semantics of RL, the judgement $\{0\}\ m\ \{0\} \vdash \{1\}\ m\ \{1\}$ is not valid, nor is $\{0 \vee 1\}\ m\ \{0\} \vdash \{1\}\ m\ \{1\}$, nor similar EL judgements, so by Corollary 8.3 they are not derivable.

9 Discussion

Banerjee et al. [BNR08] introduce a logic, essentially RL, in which command effects are specified in terms of region expressions that may depend on variables and on fields of heap objects. Their paper informally describes our second order frame rule and argues that it captures a range of separation-based disciplines for hiding module invariants in sequential object-based programs. In this paper we formalize the rule FRAME2 and the logic EL on which it is based. We show that reasoning with the rule is sound, with respect to a straightforward denotational semantics, by showing that the rule is admissible.

It is an open problem to find a semantics of EL judgements for which FRAME2 is sound as a rule. It does not seem straightforward to adapt one of the models of higher order frame rules in separation logic [BTSY05, BY07]. These capture locality properties of command meanings with respect to pre-post specifications. Our logic threads the "discipline" ($R, \overline{\eta}$ in EL judgements) through proofs so that rule RLTOEL can impose locality properties on constituent parts of client code, allowing temporary interference with the framed invariant (e.g., via rule NOUPD). Moreover our basic frame property allows shared reads and our logic validates the rule of conjunction. (Are there models of permissions [BCOP05] and higher order framing?) Perhaps it is possible to use a standard semantics and adapt the proof techniques of O'Hearn et al. [OYR04].[14] The attempt could shed light on essential similarities and differences between region logic and separation logic. A semantics for EL

[14]The full version has became available in February'08 —personal communication.

judgements would presumably entail the conditions of our forgetful semantics (Def. 7.1).

An interesting feature of FRAME2 is that the framed invariant is not required to be a "precise" predicate [OYR04] with unique semantic footprint, by contrast with the restrictions needed in separation logic in order to retain the rule of conjunctivity. On the other hand, we do not have the connective $*$ for which precision is an issue. (See discussion at the end of Sect. 8.) Like in separation logic and some versions of ownership types [MR07], objects can be transferred across encapsulation boundaries; there is no requirement that regions grow monotonically (contra, e.g., the logic of Amtoft et al. [ABB06] which inspired [BNR08]).

Previous publications by the author and others have claimed that the second order frame rule is suited only to static modules. That would be a pity. Indeed, a binary method like `equals` in Java already shows the need to rely on invariants of more than one object. To deal with dynamic allocation and invariants that pertain to small clusters of objects (such as a Collection object and its internal data structure, perhaps also shared by some Iterator objects), Parkinson and Bierman [BP05] do not use second order framing but rather make invariants everywhere explicit, but as opaque names outside the module. The Boogie methodology [LM04] hinges on an explicit "for all objects..." form of module invariant; this led the author to see how to use second order framing with dynamic modules, as sketched in Sec. 6 (and in [BNR08]). Surely this approach can be adapted to some form of iterated separating conjunction.

Another feature of Boogie that influenced our work is that Boogie expresses encapsulation in terms of ghost state (e.g., owner pointers) and specific all-states confinement invariants. Our rule FRAME2 makes a clear distinction between a module invariant and the confinement invariant plus client effect bound used to encapsulate a module. Disciplines like Boogie and friendship [NB04] use ghost state to represent some dependencies; then they stipulate intermediate annotations which, according to meta-theorems, ensure invariance of the confinement and module invariants. By disentangling what is needed for Hoare's mismatch from specific ways of achieving it, we hope to make it possible to use module-specific disciplines, e.g., tailored for specific design patterns, without need to bake the methodology into the verification system as in current practice.

Hoare's mismatch is subverted in object-oriented programs not only by sharing of mutable state but also by the possibility of re-entrant callbacks: If an object has temporarily broken its invariant and calls another method, that could lead to a call back into the object in a state where it is unsound to assume the invariant. Our rule FRAME2 is well suited to disciplines like Boogie [LM04] that deal with reentrancy using ghost state to condition invariants on whether a call is in progress. However, our rule does not capture disciplines based on "visible state semantics" [MPHL06], which deal with re-entrancy by the rather drastic requirement that each method implementation establishes *all* invariants prior to invoking *any* method.

Drossopoulou et al [DFM08] introduce a general framework to describe verification techniques for invariants based on visible state semantics. A general result shows that certain constraints on the framework parameters are sufficient for soundness. A number of disciplines from the literature are studied as instances of the framework, which promises great improvement in comparing and assessing disciplines. In its present form,

the framework does not encompass disciplines like Boogie that rely on assertions rather than types.

Kassios' technique of explicit footprints is used by Smans et al [SJPS08] to specify read effects of pure methods.

By contrast with Kassios' refinement calculus, we pay a price for formalizing a Hoare logic with separate pre/postconditions and modifies clause: for completeness, we need rules like NoUpd and FrPost to connect the different parts, as well as more use of ghost variables and effect subsumption, and a relational style loop rule.

Because regions are state-dependent, one command can interfere with the effect of another. So region logic [BNR08] uses subeffects and immunity to connect the effect of a command to the effect of its subcommands, taming the problem somewhat by following separation logic in restricting commands and assertions to read only one step into the heap, with controlled two-step reading in the case of region expressions. Kassios addresses the issue through refinement rules (cf. Metatheorem 5.4.1 in his thesis). Even a one-object effect like "writes $x.g.f$" depends on the state of the pivot $x.g$, and the difficulty is in evidence in Leino and Nelson's discussion of their "swinging pivots restriction" [LN02, Sec. 8.3].

The cited disciplines address inheritance and dynamically dispatched method calls. I plan to extend the logic to cover a core Java/JML fragment [LNR06] for which a theory of behavioral subtyping with invariants has been developed. Banerjee and others are experimenting with region logic in an SMT-based verifier and our ultimate aim is to build a verifier with its disciplines explicit and verified. Once more experience is gained with examples, we will tackle the completeness of the logic.

Thanks: to Anindya Banerjee, anonymous LICS reviewers, and participants at Dagstuhl Seminar 08061 on "Types, Logics and Semantics for State" for helpful suggestions.

References

[ABB06] T. Amtoft, S. Bandhakavi, and A. Banerjee. A logic for information flow in object-oriented programs. In *ACM Symposium on Principles of Programming Languages (POPL)*, 2006. Extended version available as KSU CIS-TR-2005-1.

[BCOP05] Richard Bornat, Cristiano Calcagno, Peter W. O'Hearn, and Matthew J. Parkinson. Permission accounting in separation logic. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 259–270, 2005.

[BNR08] Anindya Banerjee, David A. Naumann, and Stan Rosenberg. Regional logic for local reasoning about global invariants. In *European Conference on Object Oriented Programming (ECOOP)*, 2008. To appear.

[BP05] G.M. Bierman and M.J. Parkinson. Separation logic and abstraction. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 247–258, 2005.

[BTSY05] Lars Birkedal, Noah Torp-Smith, and Hongseok Yang. Semantics of separation-logic typing and higher-order frame rules. In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 260–269, 2005.

[BY07] Lars Birkedal and Hongseok Yang. Relational parametricity and separation logic. In *Foundations of Software Science and Computational Structures, 10th International Conference (FoSSaCS)*, LNCS, pages 93–107, 2007.

[DFM08] S. Drossopoulou, A. Francalanza, and P. Müller. A unified framework for verification techniques for object invariants. In *FOOL workshop*, 2008.

[GPV06] Christian Grothoff, Jens Palsberg, and Jan Vitek. Encapsulating objects with confined types. *ACM Transactions on Programming Languages and Systems*, 29(6), 2006.

[Heh84] Eric C. R. Hehner. Predicative programming part I. *Communications of the ACM*, 27:134–143, 1984.

[Hoa72] C. A. R. Hoare. Proofs of correctness of data representations. *Acta Informatica*, 1:271–281, 1972.

[Kas06] Ioannis T. Kassios. Dynamic framing: Support for framing, dependencies and sharing without restriction. In *Formal Methods: International Conference of Formal Methods Europe*, volume 4085 of *LNCS*, pages 268–283, 2006.

[LLM07] Gary T. Leavens, K. Rustan M. Leino, and Peter Müller. Specification and verification challenges for sequential object-oriented programs. *Formal Aspects of Computing*, 19(2):159–189, 2007.

[LM04] K. Rustan M. Leino and Peter Müller. Object invariants in dynamic contexts. In *European Conference on Object-Oriented Programming (ECOOP)*, pages 491–516, 2004.

[LN02] K. Rustan M. Leino and Greg Nelson. Data abstraction and information hiding. *ACM Transactions on Programming Languages and Systems*, 24(5):491–553, 2002.

[LNR06] Gary T. Leavens, David A. Naumann, and Stan Rosenberg. Preliminary definition of core JML. Technical Report CS Report 2006-07, Stevens Institute of Technology, 2006.

[MPHL06] P. Müller, A. Poetzsch-Heffter, and G. T. Leavens. Modular invariants for layered object structures. *Science of Computer Programming*, 62(3):253–286, 2006.

[MR07] P. Müller and A. Rudich. Ownership transfer in Universe Types. In *ACM Conf. on Object-Oriented Programming Languages, Systems, and Applications (OOPSLA)*, 2007.

[NB04] David A. Naumann and Mike Barnett. Towards imperative modules: Reasoning about invariants and sharing of mutable state (extended abstract). In *IEEE Symp. on Logic in Computer Science (LICS)*, pages 313–323, 2004.

[OYR04] P.W. O'Hearn, H. Yang, and J.C. Reynolds. Separation and information hiding. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 268–280, 2004.

[PBNM08] R.L. Petersen, L. Birkedal, A. Nanevski, and G. Morrisett. A realizability model of impredicative Hoare Type Theory. In *European Symposium on Programming (ESOP)*, 2008. To appear.

[SJPS08] Jan Smans, Bart Jacobs, Frank Piessens, and Wolfram Schulte. An automatic verifier for Java-like programs based on dynamic frames. In *FASE*, 2008. To appear.

[TT94] Mads Tofte and Jean-Pierre Talpin. Implementation of the typed call-by-value lambda-calculus using a stack of regions. In *ACM Symposium on Principles of Programming Languages (POPL)*, pages 188–201, 1994.

$$\textsc{Seq0}^* \quad \frac{\vdash_{R;\overline{\eta}} \{P\}\ C_1\ \{P_1\}[\overline{\varepsilon}_1; \overline{\varepsilon}_1^*] \qquad \vdash_{R;\overline{\eta}} \{P_1\}\ C_2\ \{P'\}[\overline{\varepsilon}_2; \overline{\varepsilon}_2^*] \qquad (\overline{\varepsilon}_1, \overline{\varepsilon}_1^*) \text{ is } \mathbf{fr}\text{-free} \qquad (\overline{\varepsilon}_2, \overline{\varepsilon}_2^*) \text{ is } P/(\overline{\varepsilon}_1, \overline{\varepsilon}_1^*)\text{-immune}}{\vdash_{R;\overline{\eta}} \{P\}\ C_1\ ;\ C_2\ \{P'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2;\ \overline{\varepsilon}_1^*, \overline{\varepsilon}_2^*]}$$

$$\textsc{If}^* \quad \frac{\vdash_{R;\overline{\eta}} \{P \wedge x \neq 0\}\ C_1\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad \vdash_{R;\overline{\eta}} \{P \wedge x = 0\}\ C_2\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}{\vdash_{R;\overline{\eta}} \{P\}\ \mathbf{if}\ x\ \mathbf{then}\ C_1\ \mathbf{else}\ C_2\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Conseq}^* \quad \frac{\vdash_{R;\overline{\eta}} \{P_1\}\ C\ \{P_1'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad P_2 \Rightarrow P_1 \qquad P_1' \Rightarrow P_2'}{\vdash_{R;\overline{\eta}} \{P_2\}\ C\ \{P_2'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Context}^* \quad \frac{\vdash^{\Gamma}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}{\vdash^{\Gamma, x:T}_{R;\overline{\eta}} \{P\}\ C\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

$$\textsc{Aux}^* \quad \frac{\vdash_{R;\overline{\eta}} \{P\}\ \mathbf{var}\ \overline{v}\colon \overline{T}\ \mathbf{in}\ C\ \mathbf{end}\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*] \qquad \mathit{auxil}(\overline{v}, C)}{\vdash_{R;\overline{\eta}} \{P\}\ \mathit{erase}(\overline{v}, C)\ \{P'\}[\overline{\varepsilon}; \overline{\varepsilon}^*]}$$

Figure 9: Remaining correctness rules for EL, omitting assumptions $\Delta; \Delta^*$ which are the same throughout.

A Appendix on the extended logic EL

Fig. 9 gives the remaining correctness rules for EL. The rule SEQ0* corresponds not to SEQ but to a simpler rule,

$$\textsc{Seq0} \quad \frac{\{P\}\ C_1\ \{P_1\}[\overline{\varepsilon}_1] \qquad \{P_1\}\ C_2\ \{P'\}[\overline{\varepsilon}_2] \qquad \overline{\varepsilon}_1 \text{ is } \mathbf{fr}\text{-free} \qquad \overline{\varepsilon}_2 \text{ is } P/\overline{\varepsilon}_1\text{-immune}}{\{P\}\ C_1\ ;\ C_2\ \{P'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2]}$$

that can be derived from SEQ using SUBEFF. We use this form of SEQ0* in the proofs, to lighten notation. The general form that corresponds to SEQ poses no further difficulty. But there is an interesting design choice for its formulation, similar to the following design choice for SUBEFF*. My first guess for SUBEFF* was to use separate side conditions $P \vdash \overline{\varepsilon} \leq \overline{\varepsilon}'$ and $P \vdash \overline{\varepsilon}^* \leq \overline{\varepsilon}'^*$ since in examples one finds the effects are about different parts of the state. And this does suffice to prove all the results in the sequel; in particular, the requisite connection with SUBEFF holds because together the conditions imply $P \vdash \overline{\varepsilon}, \overline{\varepsilon}^* \leq \overline{\varepsilon}', \overline{\varepsilon}'^*$. Using the latter in the rule allows "abuse", i.e., moving client effects to the module side and vice versa. But this will not lead to a successful proof using FRAME2, and anyway the rule is sound owing to Def. 7.1.

We omit EL versions of rules NOUPD and FRPOST; they pose no difficulty.

Lemma 7.2 (EL rule soundness) Aside from FRAME2, every rule of EL is sound: if its antecedents (if any) are valid and the side conditions hold then the consequent is

valid.[15]

Proof: For rules of RL, this follows directly from Prop. 5.1. Soundness of rule RLtoEL is an immediate consequence of the semantics of the judgements involved. For the rules of Figs. 8 and 9, soundness can be shown by using the definition of validity and appealing to soundness of the corresponding rule in RL. We consider Call* and Seq0* as examples. For the first, suppose Δ^* contains $(\overline{y}{:}\,\overline{U})\{P\}m(\overline{x}{:}\,\overline{T})\{P'\}[\overline{\varepsilon}]$ and let $Q \equiv P/\overline{x}{\to}\overline{z}$, $Q' \equiv P/\overline{x}{\to}\overline{z}$, $\overline{\varepsilon}' \equiv \overline{\varepsilon}/\overline{x}{\to}\overline{z}$, so that Call* yields $\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{Q\}m(\overline{z})\{Q'\}[\varnothing;\overline{\varepsilon}']$. By definition, this is valid just if $\Delta, \Delta^* \vdash \{Q\}\ m(\overline{z})\ \{Q'\}[\overline{\varepsilon}']$ is valid —and this is the consequent in rule Call, which is sound by Prop 5.1.

For rule Seq0*, suppose we have valid antecedents $\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C_1\ \{P_1\}[\overline{\varepsilon}_1; \overline{\varepsilon}_1^*]$ and $\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P_1\}\ C_2\ \{P'\}[\overline{\varepsilon}_2; \overline{\varepsilon}_2^*]$, and moreover $(\overline{\varepsilon}_1, \overline{\varepsilon}_1^*)$ is **fr**-free and $(\overline{\varepsilon}_2, \overline{\varepsilon}_2^*)$ is $P/(\overline{\varepsilon}_1, \overline{\varepsilon}_1^*)$-immune. By definition of validity, the RL judgements $\Delta, \Delta^* \vdash \{P\}\ C_1\ \{P_1\}[\overline{\varepsilon}_1; \overline{\varepsilon}_1^*]$ and $\Delta, \Delta^* \vdash \{P_1\}\ C_2\ \{P'\}[\overline{\varepsilon}_2; \overline{\varepsilon}_2^*]$ are valid, and by soundness of Seq0 we have that $\Delta, \Delta^* \vdash \{P\}\ C_1\,;\,C_2\ \{P'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2;\ \overline{\varepsilon}_1^*, \overline{\varepsilon}_2^*]$ is valid. So again by definition of EL validity we have $\Delta; \Delta^* \vdash_{R;\overline{\eta}} \{P\}\ C_1\,;\,C_2\ \{P'\}[\overline{\varepsilon}_1, \overline{\varepsilon}_2;\ \overline{\varepsilon}_1^*, \overline{\varepsilon}_2^*]$. □

Lemma 8.2 (Frame2 elimination): additional proof cases For Seq0* we use the evident consequence property of immunity and for SubEff* we use the consequence rule for sub-effecting.

[15] To be very precise we should also say all judgements are well formed.